CLAIMS

1. A multi-computer system having provision for global synchronization objects comprising:

a plurality of multi-processor nodes each having provision for local memory, threads, and an operating system having the ability to manage local synchronization objects;

global memory accessible to the processors on all the nodes and having at least one spinlock;

a data structure in memory accessible by all the processors wherein one or more records for global synchronization objects may be established, said data structure including provision for recording in a queue the identity of nodes having threads awaiting access to the synchronization object; and

a synchronization software system of programs established in all the nodes which, at the request of a thread running on a node, can create, open, request, release, and close a global synchronization object, using the above spinlock and data structure and queue of node identities to resolve requests for the synchronization object as between threads residing on different nodes, and using local synchronization objects created by the local operating systems on nodes having threads awaiting access to resolve requests for the synchronization object between threads residing on the same node.

2. A multi-computer system in accordance with claim 1 wherein the queue in which is recorded the identity of the nodes having threads awaiting access to the global synchronization object is organized as a FIFO arrangement of the node identifiers ordered in the same order in which requests for the global synchronization object are received from the threads.

3. A multi-computer system in accordance with claim 2 wherein node identifiers are moved to the end of the queue each time one of the threads on the correspondingly identified node gains ownership of the local and global synchronization objects.

4. A multi-computer system in accordance with claim 3 wherein counts are maintained for each node of the number of threads awaiting a synchronization object, wherein those counts are decremented when a thread on the corresponding

node is granted the synchronization object, and wherein the reference to the name of the corresponding node in the data structure is removed when the count reaches zero.

5. A multi-computer system in accordance with claim 2 wherein counts are maintained for each node of the number of threads awaiting a synchronization object, wherein those counts are decremented when a thread on the corresponding node is granted the synchronization object, and wherein the reference to the name of the corresponding node in the data structure is removed when the count reaches zero.

6. A multi-computer system in accordance with claim 1 wherein counts are maintained for each node of the number of threads awaiting a synchronization object, wherein those counts are decremented when a thread on the corresponding node is granted the synchronization object, and wherein the reference to the name of the corresponding node in the data structure is removed when the count reaches zero.

7. A multi-computer system in accordance with claim 1 wherein the global synchronization objects are semaphores.

8. A multi-computer system having provision for global mutexes comprising:

a plurality of multi-processor nodes each having provision for local memory, threads, and an operating system having the ability to manage local synchronization objects;

global memory accessible to the processors on all the nodes and having at least one spinlock;

a data structure in memory accessible by all the processors wherein one or more records for global mutexes may be established, said data structure including provision for recording in a queue the identity of nodes having threads awaiting access to the synchronization object; and

a synchronization software system of programs established in all the nodes which, at the request of a thread running on a node, can create, open, request, release, and close a global mutex, using the above spinlock and data structure and queue of node identities to resolve requests for the global mutex as between threads residing on different nodes, and using local mutexes created by the local operating systems on nodes having threads awaiting access to resolve requests for the global mutex between the threads residing on the same node.

9. A multi-computer system in accordance with claim 8 wherein the queue in which is recorded the identity of the nodes having threads awaiting access to the global mutex is organized as a FIFO arrangement of the node identifiers ordered in the same order in which requests for the global mutex are received from the threads.

10. A multi-computer system in accordance with claim 9 wherein the node identifiers are moved to the end of the queue each time one of the threads on the correspondingly identified node gains ownership of the local and global mutexes.

11. A multi-computer system in accordance with claim 10 wherein counts are maintained for each node of the number of threads awaiting a mutex, wherein those counts are decremented when a thread on the corresponding node is granted the mutex, and wherein the reference to the name of the corresponding node in the data structure is removed when the count reaches zero.

12. A multi-computer system in accordance with claim 9 wherein counts are maintained for each node of the number of threads awaiting a mutex, wherein those counts are decremented when a thread on the corresponding node is granted the mutex, and wherein the reference to the name of the corresponding node in the data structure is removed when the count reaches zero.

13. A multi-computer system in accordance with claim 8 wherein counts are maintained for each node of the number of threads awaiting a mutex, wherein those counts are decremented when a thread on the corresponding node is granted the mutex, and wherein the reference to the name of the corresponding node in the data structure is removed when the count reaches zero.

14. A method for granting threads running on various multi-processor nodes within a multi-computer system ownership of a global synchronization object comprising the steps of:

maintaining a record of the state of the global synchronization object as free, owned, or in transition;

when a thread seeks ownership of the global synchronization object, granting the thread, through a spinlock mechanism, access to the status of the global synchronization object, and granting the thread ownership if the object is free;

if the object is not free (owned or in transition), adding the thread's node to a queue of nodes having threads awaiting ownership of the global synchronization object, and permitting the thread to seek ownership of a local synchronization object established on the thread's node by a local operating system, but temporarily blocking threads on the thread's node from seeking ownership of the local synchronization object and forcing them into suspension;

when the global synchronization object ownership is released by a thread, placing the global synchronization object into its transition state, and then arranging for each node in the queue, in turn, to stop blocking threads on its node from seeking ownership of the local synchronization object, and permitting any thread that then gains ownership of its local synchronization object to resume execution and to gain ownership of the global synchronization object if the object is not owned (free or in transition), this process continuing until the global synchronization object is owned or until no more threads seek its ownership, at which point the global synchronization object enters its free state.

15. A method in accordance with claim 13 wherein the synchronization objects are mutexes.

16. A method in accordance with claim 13 wherein the synchronization objects are semaphores.

17. A set of synchronization software computer programs designed for use in conjunction with a multi-computer system, where individual nodes have their own copies of an operating system with local node synchronization software included in the operating system, said synchronization software computer programs being capable of carrying out the following steps to implement global synchronization objects:

maintaining a record of the state of each global synchronization object as free, owned, or in transition;

when a thread seeks ownership of a global synchronization object, granting the thread, through a spinlock mechanism, access to the status of the global synchronization object, and granting the thread ownership if the object is free;

if the object is not free (owned or in transition), adding the thread's node to a queue of nodes having threads awaiting ownership of the global synchronization object, and permitting the thread to seek ownership of a local synchronization object

established on the thread's node by a local operating system, but temporarily blocking threads on the thread's node from seeking ownership of the local synchronization object and forcing them into suspension;

when the global synchronization object ownership is released by a thread, placing the global synchronization object into its transition state, and then arranging for each node in the queue, in turn, to stop blocking threads on its node from seeking ownership of the local synchronization object, and permitting any thread that then gains ownership of its local synchronization object to resume execution and to gain ownership of the global synchronization object if the object is not owned (free or in transition), this process continuing until the global synchronization object is owned or until no more threads seek its ownership, at which point the global synchronization object enters its free state.

18. A method in accordance with claim 13 wherein the synchronization objects are mutexes.

19. A method in accordance with claim 13 wherein the synchronization objects are semaphores.